Vlerick Leuven Gent Management School
the Autonomous Management School of Ghent University and Katholieke Universiteit Leuven

Vlerick Leuven Gent Working Paper Series 2003/25

A HYBRID SCATTER SEARCH / ELECTROMAGNETISM META-HEURISTIC FOR PROJECT SCHEDULING

DIETER DEBELS

BERT DE REYCK

ROEL LEUS

MARIO VANHOUCKE

Mario.Vanhoucke@vlerick.be

A HYBRID SCATTER SEARCH / ELECTROMAGNETISM META-HEURISTIC FOR PROJECT SCHEDULING

DIETER DEBELS
Faculty of Economics and Business Administration,
Ghent University
BERT DE REYCK
London Business School
ROEL LEUS
Department of Applied Economics,
KU Leuven
MARIO VANHOUCKE
Vlerick Leuven Gent Management School

Contact
Prof Dr Mario Vanhoucke
Vlerick Leuven Gent Management School
Reep 1, 9000 Gent, Belgium
Tel: ++32 9 210 97 81
Fax:++32 9 210 97 00
E-mail: Mario.Vanhoucke@vlerick.be

ABSTRACT

In the last few decades, several effective algorithms for solving the resource-constrained project scheduling problem have been proposed. However, the challenging nature of this problem, summarised in its strongly *NP*-hard status, restricts the effectiveness of exact optimisation to relatively small instances. In this paper, we present a new meta-heuristic for this problem, able to provide near-optimal heuristic solutions. The procedure combines elements from scatter search, a generic population-based evolutionary search method, and a recently introduced heuristic method for the optimisation of unconstrained continuous functions based on an analogy with electromagnetism theory, hereafter referred to as the electromagnetism meta-heuristic. We present computational experiments on standard benchmark datasets, compare the results with current state-of-the-art heuristics, and show that the procedure is capable of producing consistently good results for challenging instances of the resource-constrained project scheduling problem. We also demonstrate that the algorithm outperforms state-of-the-art existing heuristics.

Keywords: project scheduling; heuristics; scatter search; electromagnetism

1. INTRODUCTION

We study the resource-constrained project scheduling problem (RCPSP), denoted as $m,1|cpm|C_{max}$ using the classification scheme of Herroelen et al. (1998a). The RCPSP can be stated as follows. A set of activities N, numbered from 0 to n ($|N|=n+1$), is to be scheduled without pre-emption on a set R of renewable resource types. Activity i has a deterministic duration $d_i \in \mathbb{N}$ and requires $r_{ik} \in \mathbb{N}$ units of resource type k, $k \in R$, which has a constant availability a_k throughout the project horizon. We assume that $r_{ik} \leq a_k$, $i \in N$, $k \in R$. The dummy start and end activities 0 and n have zero duration while the other activities have a non-zero duration; the dummies also have zero resource usage. A is the set of pairs of activities between which a finish-start precedence relationship with time lag 0 exists. We assume graph $G(N,A)$ to be acyclic. A schedule S is defined by an $(n+1)$-vector of start times $\mathbf{s} = (s_0,...,s_n)$ which implies an $(n+1)$-vector of finish times $\mathbf{e}$ ($e_i=s_i+d_i$, $\forall i \in N$). A schedule is said to be feasible if the precedence and resource constraints are satisfied. The objective of the RCPSP is to find a feasible schedule such that the schedule makespan e_n is minimised.

The research on the RCPSP has widely expanded over the last few decades, and reviews can be found in Brucker et al. (1999), Herroelen et al. (1998b), Icmeli et al. (1993), Kolisch and Padman (2001) and Özdamar and Ulusoy (1995). Numerous exact solution approaches have been advanced, with Brucker et al. (1998), Demeulemeester and Herroelen (1992, 1997), Mingozzi et al. (1998) and Sprecher (2000) perhaps the most noteworthy. However, the RCPSP, being a generalisation of the job shop scheduling problem, is strongly *NP*-hard (Blazewicz et al. 1983), and the computation times for exact algorithms can be excessive even for moderately sized instances. This has motivated numerous researchers to develop heuristic methods for dealing with RCPSP-instances of practical sizes. Kolisch and Hartmann (1999) and Hartmann and Kolisch (2000) present a classification and performance evaluation of different such heuristics. Additional recent sources include Alcaraz and Maroto (2001), Bouleimen and Lecocq (2003), Fleszar and Hindi (2004), Hartmann (1998, 2002), Nonobe and Ibaraki (2002), Palpant (2001), Palpant et al. (2002), and Valls et al. (2001, 2003).

In this paper, we describe a new heuristic for the RCPSP, inspired by recent advances in the development of meta-heuristics. The procedure combines elements from scatter search (SS), a population-based evolutionary search method, and a recently introduced heuristic method for the optimisation of unconstrained continuous functions that simulates the electromagnetism theory of physics, hereafter referred to as the electromagnetism (EM) meta-

heuristic (Birbil and Fang 2003). We extend the EM heuristic for combinatorial optimisation problems and integrate it in a scatter search framework. In Section 2, we describe the main elements of the EM heuristic applied to unconstrained continuous optimisation. Section 3 discusses how we represent and evaluate RCPSP solutions to be used in a scatter search framework. In Section 4, we show how the EM methodology can be modified and enhanced to be used in a combinatorial optimisation setting and how it can be integrated with scatter search for the RCPSP. Section 5 and 6 discuss intensification and diversification strategies employed to enhance the effectiveness and efficiency of the algorithm. Section 7 contains the computational results on benchmark datasets, as well as a comparison with other current state-of-the-art heuristics. We conclude with Section 8.

2. THE ELECTROMAGNETISM META-HEURISTIC

Birbil and Fang (2003) propose a so-called electromagnetism (EM) optimisation heuristic for unconstrained global optimisation problems, i.e. the minimisation of non-linear functions. In a multi-dimensional solution space where each point represents a solution, a *charge* is associated with each point. This charge is related to the objective function value associated with the solution point. As in evolutionary search algorithms, a population, or set of solutions, is created, in which each solution point will exert attraction or repulsion on other points, the magnitude of which is proportional to the product of the charges and inversely proportional to the distance between the points (Coulomb's Law). The principle behind the algorithm is that inferior solution points will prevent a move in their direction by repelling other solution points in the population, and that attractive points will facilitate moves in their direction. This can be seen as a form of local search in Euclidian space in a population-based framework, similar to scatter search. The main difference with existing methods is that the moves are governed by forces that obey the rules of electromagnetism. Birbil and Fang (2003) provide a generic pseudo-code for the EM algorithm:

```
Algorithm EM(maxiter, LSiter)
    iter :=1
    while iter<maxiter do
        local(LSiter)
        compute_forces
        apply_forces
        iter++
    endwhile
```

The function EM requires two parameters, *maxiter* defining the number of iterations or populations, and *LSiter* defining the number of iterations in a local search sub-procedure *local*, which is applied before a new population is created. The function *local* explores the immediate (Euclidian) neighbourhood of individual points. The total force exerted on each point by all other points is calculated in function *compute_forces*, which depends on the charge of the point under consideration as well as of the points exerting the force. The charge of each point $\mathbf{x}_i$ is determined by its objective function value $f(\mathbf{x}_i)$ in relation to the objection function value of the current best point $\mathbf{x}^{best}$ in the population, with better objective function values resulting in higher charges. For a minimisation problem, the charge q_i of particle $\mathbf{x}_i$ is determined according to equation (2.1).

$$q_i = \exp\left(-d\frac{f(\boldsymbol{x}_i)-f(\boldsymbol{x}^{best})}{\sum_{k=1}^{m}(f(\boldsymbol{x}_k)-f(\boldsymbol{x}^{best}))}\right) \tag{2.1}$$

The parameter m represents the population size, d is the dimension of the solution space. In the function *compute_forces*, a set of force vectors $\mathbf{F}_i$ is determined, i=1,…,m, that are exerted on particle $\mathbf{x}_i$:

$$\mathbf{F}_i = \sum_{\substack{j=1\\ j\neq i}}^{j=m} \left\{ \begin{array}{ll} (\boldsymbol{x}_j-\boldsymbol{x}_i)\left(\frac{q_i q_j}{\|\mathbf{x}_j-\mathbf{x}_i\|^2}\right) & \text{if } f(\boldsymbol{x}_j) < f(\boldsymbol{x}_i) \\ (\boldsymbol{x}_i-\boldsymbol{x}_j)\left(\frac{q_i q_j}{\|\mathbf{x}_j-\mathbf{x}_i\|^2}\right) & \text{if } f(\boldsymbol{x}_j) \geq f(\boldsymbol{x}_i) \end{array} \right\} \tag{2.2}$$

The point with a relatively good objective function value attracts the other one, the point with the inferior objective value repels the other. The forces exerted on i by each of the other points are combined by means of vector summation, as shown in the example in Figure 1. In the example, $\mathbf{F}_{13}$ is the force exerted by $\mathbf{x}_1$ on $\mathbf{x}_3$ (repulsion: the objective function value of $\mathbf{x}_3$ is better than that of $\mathbf{x}_1$) and $\mathbf{F}_{23}$ is the force exerted by $\mathbf{x}_2$ on $\mathbf{x}_3$ (attraction: the objective function value of $\mathbf{x}_3$ is worse than that of $\mathbf{x}_2$). The total force exerted on $\mathbf{x}_3$ equals $\mathbf{F}_3=\mathbf{F}_{13}+\mathbf{F}_{23}$.

Insert Figure 1 About Here

The movement according to the resulting forces is performed in *apply_forces*, which generates a new population. Contrary to the simplified example in Figure 1, the imposed force is normalised, by dividing it by its norm, and therefore only identifies the direction of the move, not the magnitude. The magnitude of each move is determined for each dimension separately, and is equal to a value randomly selected from domain [0; *maxmove*], where *maxmove* indicates the maximum allowable move in the particular dimension.

Birbil and Fang (2003) use *maxiter* iterations, or populations, although other termination criteria could be applied, such as a maximum number of iterations without improving the current best solution. Convergence details for the heuristic are provided in Birbil et al. (2003).

EM is a 'population based' or 'evolutionary' algorithm, since it operates on a population of solutions rather than on a single solution at a time. This makes it most closely related to genetic algorithms (GA, Goldberg 1989) and to scatter search (SS, Glover et al. 2003). SS, being a generic methodology, can be seen as a generalisation of the GA procedure (Taillard et al. 2001). SS is a population-based method where new solutions are constructed using convex or non-convex linear combinations of solutions. In our procedure, the selection and combination rules for SS are provided by the EM framework, which, not unlike GA, can be seen as a particular form of the SS algorithm. In the following section, we will discuss how we have extended the EM methodology for combinatorial optimisation and the RCPSP in particular, and how it can be integrated in a general SS framework.

3. REPRESENTATION, SCHEDULE GENERATION AND SOLUTION EVALUATION

The backbone of most improvement heuristics for solving the RCPSP, where an initial (set of) solution(s) is gradually improved, is a schedule representation scheme, a schedule generation scheme and a solution evaluation procedure. Typically, an RCPSP improvement heuristic does not operate directly on a schedule, but on some representation of a schedule that is convenient and effective for the functioning of the algorithm. After an operation on a solution (i.e. on a schedule represented in a particular way) has been performed, the newly obtained solution is transformed into a schedule using a schedule generation scheme (SGS).

Kolisch and Hartmann (1999) distinguish between various representations for schedules in the development of heuristics for the RCPSP. The two most important ones are the *random-key (RK)* representation and the *activity-list (AL)* representation. In RK form, a

solution corresponds to a point in Euclidian (n+1)-space, such that the i-th vector element functions as a priority value for the i-th activity. Using a serial schedule-generating scheme, these priority values can then be used to construct an active schedule by scheduling each activity one-at-a-time and as soon as possible within the precedence and resource constraints. Alternatively, a parallel SGS could be used, although Kolisch (1996) has shown that, contrary to the serial SGS, the parallel SGS is sometimes unable to reach an optimal solution. In the AL representation, a schedule is represented by a linear extension of the partial order induced by the precedence constraints, such that a SGS gives priority to the activity that comes first in the list containing a complete order on N. This is similar to list scheduling in machine scheduling.

Hartmann and Kolisch (2000) report that in general, procedures that make use of the AL representation perform better than those based on the RK form. This claim is based solely on computational tests, and no underlying reasons are cited. We believe that the main reason for the inferior performance of the RK representation lies in the fact that one single schedule can have many different representations. This results in a larger solution space, and the issue that the structure of a solution or schedule representation does not necessarily contain information about the quality of the associated schedule, which sometimes prevent (meta-) heuristics operating on schedule representations from making improvements. The AL representation also suffers from this, in that a single schedule can be represented by different activity lists. This problem, however, occurs more frequently using the RK representation, for reasons we will explain below.

The RK representation, however, has the advantage that each solution corresponds to a point in Euclidian (n+1)-space, so that geometric operations can be performed on its components. Since this is one of the cornerstones of both the SS and EM methods, we adopt the RK representation, allowing us to perform mathematical operations on solutions. We have modified the standard RK representation in order to eliminate the problem stated above, thereby removing its comparative disadvantage relative to the AL form.

There are four underlying reasons why a schedule can be represented by different RK forms, caused by (1) scaling, (2) precedence constraints, (3) timing anomalies and (4) activities with identical starting times. We will discuss these problems one by one and show how these problems can be eliminated using a unique, standardised form of the RK representation. Note that problems (3) and (4) also occur for activity lists. By eliminating all four problem areas, our unique RK representation will perform better than both the standard RK as well as the standard AL forms.

We introduce the example project depicted in Figure 2, with a single renewable resource type with availability a_1=2. A feasible schedule for this scheduling problem, with a makespan equal to 18, is given in Figure 3. Assuming that lower RK values correspond to higher priorities, the schedule in Figure 3 can be obtained with the following RK vector: $\mathbf{x}_1$= [0.9; 1.1; 2.6; 2.9; 2.1; 3.5; 0.7; 1.9; 3.2] (we omit the RK values for the dummy start and end activity).

Insert Figure 2 & 3 About Here

(1) *Scaling in Euclidean space*

Scaling the priority values of any RK representation up or down results in a different RK representation, which, however, will always result in the same schedule. In fact, there exist an infinite number of RK representations with different priority values, but with the same priority structure. For our example, $\mathbf{x}_2$= [8; 13; 31; 32; 26; 52; 3; 17; 48] results in the same schedule. We eliminate this problem by replacing the priority values by their rank values. For the example, we can transform $\mathbf{x}_1$ or $\mathbf{x}_2$ into $\mathbf{x}_3$= [2; 3; 6; 7; 5; 9; 1; 4; 8], which also yields the schedule in Figure 3.

(2) *Precedence constraints*

In an RK representation, priority values are not constrained by the precedence constraints, in the sense that the RK of an activity can be higher than the RK of one of its predecessors. Essentially, this is not a problem since a SGS will take the precedence relations into account, but it can lead to different RK representations for a single schedule. In our example, we can see in $\mathbf{x}_3$, that activity 7 has a higher priority (a lower RK) than activities 1, 2, 3 and 4, the predecessors of activity 7. A serial SGS would schedule the activities in the following order: 1, 2, 8, 5, 3, 4, 6, 7 and finally 9, i.e. taking into account the precedence relations. Another RK representation such as $\mathbf{x}_4$= [1; 2; 6; 7; 5; 9; 3; 4; 8] would result in the same schedule. To eliminate this problem, we set the RK values of each activities equal to their rank order in the activity list obtained using a serial SGS. This results in an RK representation with priority values “in line” with the precedence constraints. For our example, we obtain $\mathbf{x}_5$= [1; 2; 5; 6; 4; 7; 8; 3; 9].

(3) *Timing anomalies*

The previous two problems arise only with the RK representation. There are two more problems, associated with both the RK and AL representation. The first is caused by the following phenomenon. If an activity a_1 starts earlier than another activity a_2 in a schedule, then an AL representation of this schedule exists with a_1 before a_2. If, however, none of the activities starting after a_1 and before a_2 in the activity list, nor a_2 itself, could be scheduled earlier if activity a_1 is removed from the schedule (because of precedence and/or resource constraints), then there also exists an AL representation for the same schedule in which a_1 right comes after a_2.

In the example schedule of Figure 3, activity 5 starts earlier than activity 8. Therefore, there is an AL in which activity 5 has higher priority than activity 8. Nevertheless, in $\mathbf{x}_5$= [1; 2; 5; 6; 4; 7; 8; 3; 9], which leads to the schedule in Figure 3, the RK of activity 5, namely 4, is higher than the RK of activity 8, namely 3, and thus activity 5 receives lower priority although its starts earlier. This is due to the fact that even in the absence of activity 5, activity 8 cannot be scheduled earlier due to activity 1 and 2 requiring a significant amount of the resource in periods 1 through 6. Activity 5, consuming less resource and taking less time, can be inserted into the schedule at time 0 both before and after activity 8 is included. In other words, there are at least two priority vectors leading to the same schedule.

To deal with this problem, we propose to use a topological-order (TO) representation of schedules (Valls et al. 1999, 2003): for a schedule S, a TO representation of S is any vector $\mathbf{x}$ containing the numbers from 0 to n+1 and for which $s_i(S) < s_j(S)$ implies $x_i < x_j$. Adhering to the TO representation eliminates the problem discussed above. For the example schedule in Figure 3, activity 5 receives the second highest priority in the TO representation. Consequently, $\mathbf{x}_5$ is replaced by $\mathbf{x}_6$ = [1; 3; 5; 6; 2; 7; 8; 4; 9].

(4) *Activities with the same starting times*

Even with the TO representation, there can still be multiple representations of a single schedule. If two activities a_1 and a_2 start at the same time, the position of a_1 and a_2 in an AL can be interchanged without affecting the associated schedule. To prevent this, we take the average of the rankings in the AL of activities starting at the same time, which will be the same for all the activities under consideration. By doing so for the example

schedule, we end up with $\mathbf{x}_7$ = [1.5; 3; 4.5; 6; 1.5; 7; 8; 4.5; 9], a unique standardised random key (SRK) representation for the schedule**.**

Using the SRK schedule representation, each population member is uniquely associated with a schedule. The EM algorithm, however, transforms the priority vectors by moving them in Euclidian space according to the (electromagnetic) forces exerted on them. These new priority vectors may again suffer from any of the four problems mentioned above. Therefore, we re-write each new priority vector in SRK-form, while at the same time evaluating the associated objective function value, as follows. When a priority vector $\mathbf{x} \in \mathbb{R}^{n+1}$ is transformed into a vector $\mathbf{y}$, we compute a schedule $S=s(\mathbf{y})$, using a SGS s, with associated objective function value equal to the makespan $e_n(s(\mathbf{y}))$. We then replace $\mathbf{x}$ by SRK priority vector $p(s(\mathbf{y}))$, where p transforms the schedule to SRK-standardised priorities, based on the activity starting times in $s(\mathbf{x})$.

4. MODIFYING THE EM ALGORITHM FOR THE RCPSP

In the basic EM algorithm, all points in a population exert a force on all other points. We generalise this concept by allowing a variable number b of points to act on any given point, with $b \in [1; m-1]$, where m is the population size. The selected set of points is referred to as B. Furthermore, we generalise the basic EM algorithm as follows. When determining the force exerted on point i by point j, we do not use fixed charge q_i and q_j to compute the attraction or repulsion force, but rather a charge q_{ji} that depends on the relative difference in objective function value between i and j. So, contrary to the basic EM algorithm, point charges are not computed independently but based on the point they exert force on:

$$q_{ji} = \frac{f(\mathbf{x}_i) - f(\mathbf{x}_j)}{f(\mathbf{x}^{worst}) - f(\mathbf{x}^{best})} \qquad (4.1)$$

with $\mathbf{x}^{worst}$ and $\mathbf{x}^{best}$ the worst and best solutions in the current generation. As a result, $q_{ji} \in [-1;1]$ and 'better' points j have higher scores on q_{ji}. More specifically, if $f(\mathbf{x}_i) > f(\mathbf{x}_j)$, i.e. when point i has higher makespan than particle j, q_{ji} is positive and particle j attracts particle i. The opposite is true when $f(\mathbf{x}_i) < f(\mathbf{x}_j)$ and repulsion occurs. No action results when $f(\mathbf{x}_i) = f(\mathbf{x}_j)$. In our implementation, forces are computed as follows:

$$\mathbf{F}_i = \sum_{\substack{j=1 \\ j \neq i}}^{B} (\boldsymbol{x}_j - \boldsymbol{x}_i) \left(\frac{c_1 q_{ji} + c_2 d_{ji}}{D(\boldsymbol{x}_i, \boldsymbol{x}_j) + c_2} \right) \tag{4.2}$$

with

$$d_{ji} = \begin{cases} 1 & \text{if } q_{ji} > 0 \\ 0 & \text{if } q_{ji} = 0 \\ -1 & \text{if } q_{ji} < 0 \end{cases} \tag{4.3}$$

c_1 and c_2 are parameters that allow to calibrate the importance of the distance measure $D(\mathbf{x}_i,\mathbf{x}_j)$, which is the sum of the absolute values of the component-wise difference between $\mathbf{x}_i$ and $\mathbf{x}_j$.

Based on the calculated forces and resulting attraction and repulsion, points are transformed, i.e. moved in Euclidian space, resulting in a new population. In the basic EM algorithm, forces are exerted in each dimension. For the RCPSP, this corresponds to a change in the priority of each activity. We generalise this concept by allowing forces to act only in a particular subset of the dimensions or activities. We randomly select $p_{\min} \in [1;n-1]$ and $p_{\max} \in [2;n]$, with $p_{\min} < p_{\max}$, and update only the RK values between $p_{\min}$ and $p_{\max}$ (inclusive) according to the forces exerted in these dimensions. Note that due to the SRK representation, the thus updated activities all start within a particular time interval. The other RK components are not left unchanged, but are updated as follows. We subtract the constant value n from all RK values lower than $p_{\min}$, and add this same constant to all values higher than $p_{\max}$. Doing this preserves the priority structure of the activities unaffected by the forces, and the relative priorities of the three resulting subsets of activities. The resulting RK vector is not in SRK form but can be transformed in SRK format.

We again consider the example project presented in Section 3, SRK-vector $\mathbf{x}_7$ = [1.5; 3; 4.5; 6; 1.5; 7; 8; 4.5; 9], p_{min}= 3 and p_{max}=6. In Table 1, the components of $\mathbf{x}_7$ in interval [$p_{\min}$; $p_{\max}$] are underlined, and the force **F** imposed on $\mathbf{x}_7$ is given. The vector added to $\mathbf{x}_7$ is referred to as **F**', and $\mathbf{x_7}+\mathbf{F}' = \boldsymbol{x}'_7$.

Insert Table 1 About Here

5. INTENSIFICATION USING LOCAL SEARCH

The makespan $e_n(s(\mathbf{x}))$ associated with a solution $\mathbf{x}$ is obtained using a serial SGS s. In order to improve the intensification characteristics of the algorithm, we use an enhanced generation scheme s^* that iteratively looks for improvements in the priority vector using forward and backward global shifts of individual activities. The scheme s^* guarantees that $e_n(s^*(\mathbf{x})) = e_n(s(\mathbf{x}))$ so that we can replace $\mathbf{x}$ by solution $p(s^*(\mathbf{x}))$. Our method is based on the basic principles described by Li and Willis (1992) and Özdamar and Ulusoy (1996).

First we apply s on $\mathbf{x}$, yielding an active, i.e. left-justified schedule. Next, we iteratively perform backward and forward passes. The priorities used in these passes are based on the RK-vector consisting of the ending times (backward) and starting times (forward) in the schedule, which results in a right-justified and left-justified schedule, respectively. It is guaranteed that the schedule makespan of each intermediate schedule is never higher than the makespan of the previous one. In effect, rather than completely rescheduling, we exploit opportunities for global right and left shifts of individual activities in order to reduce the makespan.

Insert Figure 4 About Here

A computational example will illustrate our approach: consider the project of Figure 2 and RK-vector $\mathbf{x}_1$ = [0.9; 1.1; 2.6; 2.9; 2.1; 3.5; 0.7; 1.9; 3.2]. $s(\mathbf{x}_1)$ was depicted Figure 3, which is repeated at the top of Figure 4. The schedule has a makespan of 18 time units. We now try to reduce the makespan by scheduling each activity as much as possible to the right, in decreasing order of activity end times, without affecting the project makespan. Activity 9 and 7 cannot be scheduled later. Activity 6 can be right shifted to start at time 15. Activity 4 can be shifted two time units and start at time 10. Since the global right shift of activity 6 has made some additional resources available, activity 8 can be shifted three time units to start at time instant 9. Activities 3, 2 and 1 can shift two time units. Finally, activity 5 is shifted to time 14. In this way, we obtain a schedule with a makespan of 16 units. Further improvements of the schedule are possible by shifting activities as much as possible to the left. This reduces the

makespan by one further time unit, as illustrated in Figure 4. This procedure is continued until no further improvements can be found.

As in the original EM algorithm, we can use a function *local* to improve population members in the foregoing manner. Contrary to the original algorithm, however, we perform this search immediately after new members are added to the population as a result of the function *apply_forces*, rather than at the start of each iteration.

6. DIVERSIFICATION USING MUTATION

In order to prevent the population from becoming overly homogeneous, we introduce a means of diversification using mutation, by swapping the RK values of two randomly chosen activities that are not precedence related. This mutation is imposed right after a force is executed, and only afterwards, makespan evaluation takes place. Additionally, we replace the population by new points when the makespans of the schedules in the population are identical. The initial population is also generated randomly to ensure a diversified starting solution.

7. COMPUTATIONAL EXPERIMENTS

We have coded the procedure in Visual C++ 6.0 and performed computational tests on an Acer Travelmate 634LC with a Pentium IV 1.8 GHz processor using two different testsets. The first set is composed of instances generated by *RanGen* (Demeulemeester et al. 2003) and is used to study the impact of the different parameters on the performance of the algorithm. The second testset is the well-known PSPLIB testset (Kolisch and Sprecher 1997), used to report computational results of our procedure and to compare with other state-of-the-art results.

7.1. Impact of the parameters

To test the impact of the different parameters on the effectiveness and efficiency of the procedure, we have constructed a dataset containing 480 instances using *RanGen* (Demeulemeester et al. 2003). Each instance contains 75 activities and has been generated with the following settings. The order-strength is set at 0.25, 0.50 or 0.75, resource usage at 1, 2, 3 or 4 and the resource-constrainedness at 0.2, 0.4, 0.6 or 0.8. Using 10 instances for each problem class, we obtain a problem set with 480 network instances.

This approach is similar to the way Valls et al. (2001, 2003) derive their computational results. The authors optimise the values of the different parameters based on a subset of the J120 instances, and then test the effectiveness of the algorithm on the complete testset.

Although the results would be improved by optimising the parameter values for the complete testset, the approach by Valls et al. (2001, 2003) is more suitable since the results do not rely on customising the parameters for that particular set. We opt for a similar approach, but take it one step further by not optimising the parameter values on the testset at all, not even on a subset, but on a completely different testset as described in this section.

Table 2 illustrates the influence of the size of the population m and the parameter b, i.e. the number of points exerting a force on any given point, on the performance of the algorithm. The column "Sum" contains the sum of the 480 project makespans, and the column 'Avg. Dev.' contains the average deviation from the critical path based lower-bound. The table reveals that the algorithm performs best with a population size of 8 and b equal to 1, although the algorithm seems very robust with respect to these parameter values. Similarly, optimal values for parameters c_1 and c_2 were found to be 15 and 10.

Insert Table 2 About Here

7.2. Comparative results with best known solutions

In order to compare with the best results from literature, we use the well-known J30, J60, J90 and J120 instances of the PSPLIB testset (Kolisch and Sprecher 1997). Table 3 shows the results. The row labelled "Sum" contains the sum of the makespans of all problem instances. The row labelled "Avg. Dev. CPM" reports the average deviation from the critical path based lower-bound. Since all J30 problem instances have been solved to optimality by branch-and-bound procedures from the literature, we do not report the deviation for this problem set. The row labelled "Avg. Dev. Best" displays the average percentage deviation from the current best solution in PSPLIB as reported on September 12, 2003. For the J30 set these solutions are all optimal. The fourth row, labelled "Best" shows the number of instances for which our heuristic algorithm reports a makespan equal to the current best solution. The last rows, labelled "Avg. CPU" and "Max. CPU", indicate the average and maximal computation time to solve a problem instance. Each cell of the table displays the results for a run with maximum 1,000, 5,000, 50,000 and 500,000 schedules.

Insert Table 3 About Here

The results indicate that the algorithm is capable of providing near-optimal solutions for set J30 within very small computation times, and competitive solutions for the other problems sets, all with limited computational effort. Also, the results show only a moderate increase in required computational effort when the problem size increases, which is an encouraging result since this allows the solution of very large scale instances.

7.3. Comparative results with 5,000 schedule limit

In the following tables we report a comparison with the best heuristic procedures as reported in the literature. In order to have a fair base of comparison, we only compare the results with a limit of 5,000 schedules, and omit procedures that do not report such results (these will be discussed later). To measure the effectiveness of the algorithms, we report the average deviation of the heuristic solutions from the critical path, except for J30, where we report the average deviation from the optimal solution. We also provide a rank order of effectiveness for each problem set in column "R". Empty cells denote that, to the best of our knowledge, no results have been reported in literature. Table 4 reveals that our new algorithm performs consistently well over all problem sets, and outperforms the current best-performing procedure in each class.

Insert Table 4 About Here

7.4. Comparative results with extended time limit

In this section we provide a comparison with other state-of-the-art heuristics for which computational results with a limited number of schedules are not available. These include Valls et al. (2001), Valls et al. (2003) and Fleszar and Hindi (2004). We also compare with results obtained by the algorithm of Nonobe and Ibaraki (2002) without a limit on the number of schedules (Valls 2003). Because the results for the different algorithms have been obtained using different computers, a direct comparison is not possible. Rather, we will show that our algorithm is able to outperform these heuristics with a specific limit on the number of schedules generated. As measures of algorithmic effectiveness and efficiency, we report the sum of the project makespans, the average deviations from the critical path (except for J30,

where we report the average deviation from the optimal solution) and average and maximum CPU times, where available.

Nonobe and Ibaraki (2002) developed a tabu search algorithm for RCPSP, for which new computational results are reported by Valls et al. (2003). These results, given in Table 5, show that we are able to outperform their results using only 5,000 schedules, except for J30, where we need slightly more. We therefore outperform their results with far less required computation time, even if we take into account the difference in computers (Sun Ultra 2 running at 300 MHz versus 1.8 GHz PC).

Insert Table 5 About Here

Recently, Fleszar and Hindi (2004) have developed a heuristic for the RCPSP based on variable neighbourhood search. They report good computational results, but requiring substantial computational effort. For sets J60 and J120, Fleszar and Hindi (2004) report average deviations from the critical-path lower bound of 10.94% and 33.10%. Our algorithm is capable of producing deviations of only 10.70% and 31.72% with 500,000 schedules, and 10.90% and 32.37% with 50,000 schedules, respectively. This indicates that we are outperforming their results, even with a maximum of 50,000 schedules, whereas Fleszar and Hindi (2004) do not set a limit on the number of schedules, which runs to a maximum of more than 1 million for J60 and more than 10 million for J120. They also report high computation times up to a maximum of 1,127 seconds (1 GHz processor), compared to slightly more than 15 seconds for our procedure (with 50,000 schedules on a 1.8 GHz processor). Based on these results, our results clearly outperform those of Fleszar and Hindi (2004).

Valls et al. (2003) present a heuristic based on critical activity re-ordering. Although their results for the J30 set are good, and require a 50,000 schedule-limit for our procedure to be able to outperform it, the results are rather disappointing for sets J60, J90 and J120, where our algorithm can produce better results with only 5,000 schedules. The CPU time required by Valls et al. (2003) is limited, but even considering the different processor speeds (400 MHz versus 1.8 GHz), our procedure requires even less time. This is especially clear for set J120, where Valls et al. (2003) require 17 times the CPU time we need to outperform them.

Valls et al. (2001) report excellent results, especially for sets J60, J90 and J120, as shown in Table 6. In their paper, Valls et al. (2001) show that their results outperform all other state-of-the-art heuristics, although their procedure is not subjected to a schedule limit, whereas the other procedures are. The authors show, however, that even with extended time limits, the other heuristics are not able to outperform their results. Using our new procedure, we are able to outperform these results, using 5,000 schedules for J30; 50,000 schedules for J60 and J90; and 500,000 schedules for J120. Note, however, that in order to outperform the results of Valls et al. (2001), our procedure requires more CPU time if we take into account the difference in processor speed (400 MHz versus 1.8 GHz).

8. CONCLUSIONS

In this paper, we have presented a new heuristic procedure for solving the resource-constrained project scheduling problem (RCPSP), one of the most challenging combinatorial optimisation problems in scheduling. The procedure is a population-based evolutionary method, and combines elements from scatter search and a novel method originally introduced for optimising unconstrained continuous functions based on an analogy with electromagnetism theory. We have shown how this electromagnetism heuristic can be extended for application to combinatorial optimisation problems and the RCPSP, and how it can be integrated into a scatter search framework. The procedure is equipped with intensification and diversification methods to improve its effectiveness. The computational results show that the procedure outperforms other state-of-the-art heuristics in the literature, and that it is competitive with the procedure of Valls et al. (2001), which is probably the most effective heuristic presented in the literature to date.

REFERENCES

Alcaraz, J., Maroto, C. (2001). A robust genetic algorithm for resource allocation in project scheduling. Annals of operations Research, 102, 83-109.

Birbil, S.I. and Fang, S.C. (2003). An electromagnetism-like mechanism for global optimization. Journal of Global Optimization, 25, 263-282.

Birbil, S.I., Fang, S.C. and Sheu, R.-L. (2003). On the convergence of a population-based global optimization algorithm. Journal of Global Optimization, forthcoming.

Blazewicz, J., Lenstra, J.K. and Rinnooy Kan, A.H.G. (1983). Scheduling subject to resource constraints: classification and complexity. Discrete Applied Mathematics, 5, 11-24.

Bouleimen, K. and Lecocq, H. (2003). A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version. European Journal of Operational Research, 149, 268-281.

Brucker, P., Drexl, A., Möhring, R., Neumann, K. and Pesch, E. (1999). Resource-constrained project scheduling: notation, classification, models and methods. European Journal of Operational Research, 112, 3-41.

Brucker, P., Knust, S., Schoo, A. and Thiele, O. (1998). A branch & bound algorithm for the resource-constrained project scheduling problem. European Journal of Operational Research, 107, 272-288.

Demeulemeester, E. and Herroelen, W. (1992). A branch-and-bound procedure for the multiple resource-constrained project scheduling problem. Management Science, 38, 1803-1818.

Demeulemeester, E. and Herroelen, W. (1997). New benchmark results for the resource-constrained project scheduling problem. Management Science, 43, 1485-1492.

Demeulemeester, E., Vanhoucke, M. and Herroelen, W. (2003). A random generator for activity-on-the-node networks. Journal of Scheduling, 6, 13-34.

Fleszar, K. and Hindi, K.S. (2004). Solving the resource-constrained project scheduling problem by a variable neighbourhood search. European Journal of Operational Research, forthcoming.

Glover, F., Laguna, M. and Martí, R. (2003). Scatter search. In: Ghosh, A. and Tsutsui, S. (eds.). Theory and Applications of Evolutionary Computation: Recent Trends. Springer-Verlag, forthcoming.

Goldberg, D. (1989). Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley.

Hartmann, S. (1998). A competitive genetic algorithm for the resource-constrained project scheduling. Naval Research Logistics, 45, 733-750.

Hartmann, S. (2002). A self-adaptive genetic algorithm for project scheduling under resource constraints. Naval Research Logistics, 49, 433-448.

Herroelen, W., Demeulemeester, E. and De Reyck, B. (1998a). A classification scheme for project scheduling, Chapter 1 in Weglarz, J. (Ed.), Project Scheduling – Recent Models, Algorithms and Applications. International Series in Operations Research and Management Science, 14, 77-106, Kluwer Academic Publishers.

Herroelen, W., De Reyck, B. and Demeulemeester, E. (1998b). Resource-constrained project scheduling: a survey of recent developments. Computers and Operations Research, 25 (4), 279-302.

Icmeli, O., Erenguc, S.S. and Zappe, C.J. (1993). Project scheduling problems: a survey. International Journal of Operations and Productions Management, 13 (11), 80-91.

Kolisch, R. (1996). Serial and parallel resource-constrained project scheduling methods revisited: theory and computation. European Journal of Operational Research, 43, 23-40.

Kolisch, R. and Hartmann, S. (1999). Heuristic algorithms for solving the resource-constrained project scheduling problem: classification and computational analysis. In: Weglarz, J. (ed.). Project Scheduling – Recent Models, Algorithms and Applications, 147-178, Kluwer Academic Publishers, Boston.

Kolisch, R. and Padman, R. (2001). An integrated survey of deterministic project scheduling. Omega, 49 (3), 249-272.

Kolisch, R. and Sprecher, A. (1997). PSPLIB - A project scheduling library. European Journal of Operational Research, 96, 205-216.

Li, K.Y. and Willis, R.J. (1992). An iterative scheduling technique for resource-constrained project scheduling. European Journal of Operational Research, 56, 370-379.

Mingozzi, A., Maniezzo, V., Ricciardelli, S., Bianco, L. (1998). An exact algorithm for the resource-constrained project scheduling problem based on a new mathematical formulation. Management Science, 44, 715-729.

Nonobe, K. and Ibaraki, T. (2002). Formulation and tabu search algorithm for the resource constrained project scheduling problem (RCPSP). In: Ribeiro, C.C. and Hansen, P. (Eds.). Essays and Surveys in Metaheuristics, 557-588, Kluwer Academic Publishers.

Özdamar, L. and Ulusoy, G. (1995). A survey on the resource-constrained project scheduling problem. IIE Transactions, 27, 574-586.

Özdamar, L. and Ulusoy, G. (1996). A note on an iterative forward/backward scheduling technique with reference to a procedure by Li and Willis. European Journal of Operational Research, 89, 400-407.

Palpant, M. (2001). Conception d'une métaheuristique et application au problème d'ordonnancement de project à moyens limités. Mémoire de DEA d'Informatique, http://www.lim.univ-mrs.fr/~vancan/dea/dea2001/memoires.html.

Palpant, M., Artigues, C. and Michelon, P. (2002). Solving the resource-constraint project scheduling problem by integrating exact resolution and local search. Eight International Workshop in Project Management and Scheduling, April 3-5, Valencia, Spain.

Sprecher, A. (2000). Scheduling resource-constrained projects competitively at modest resource requirements. Management Science, 46, 710-723.

Taillard, E.D., Gambardella, L.M., Gendreau, M. and Potvin, J.-Y. (2001). Adaptive memory programming: a unified view of metaheuristics. European Journal of Operational Research, 134, 1-16.

Valls, V., Ballestín, F. and Quintanilla, S. (2001). A population-based approach to the resource-constrained project scheduling problem. Technical Report TR10-2001, Departamento de Estadística e Investigación Operativa, Universidad de Valencia.

Valls, V., Quintanilla, S. and Ballestín, F. (2003). Resource-constrained project scheduling: a critical activity reordering heuristic. European Journal of Operational Research, 149, 282-301.

FIGURE 1

Example of exertion of forces

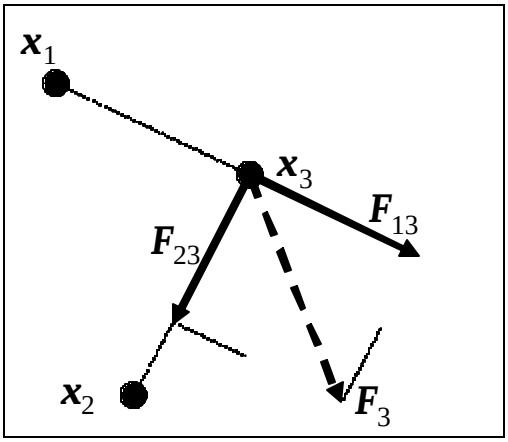

FIGURE 2

Example project

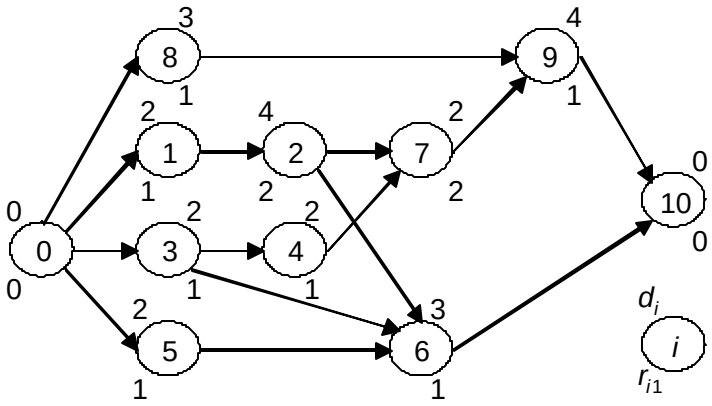

FIGURE 3

A schedule for the example project

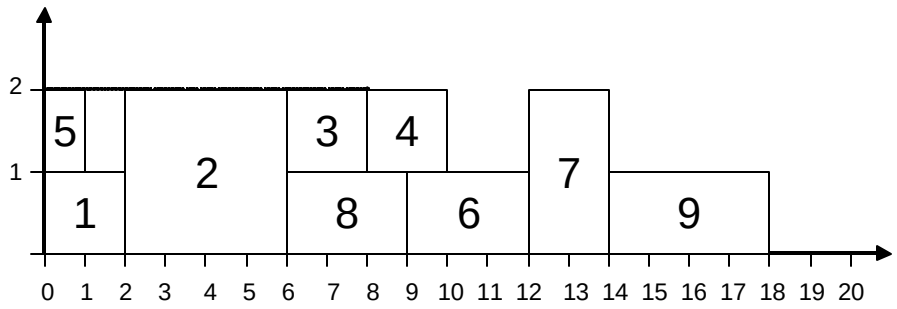

TABLE 1

Illustration of the execution of a move according to a force

Activities	1	2	3	4	5	6	7	8	9
$\mathbf{x_7}$	1.5	3	4.5	6	1.5	7	8	4.5	9
F	0.6	2.2	0.4	-0.9	0	-2.2	1.2	1.9	0.5
F'	-10	2.2	0.4	-0.9	-10	10	10	1.9	10
$\boldsymbol{x'_7}$	-8.5	5.2	4.9	5.1	-8.5	17	18	6.4	19

FIGURE 4

Stepwise improvement of the makespan

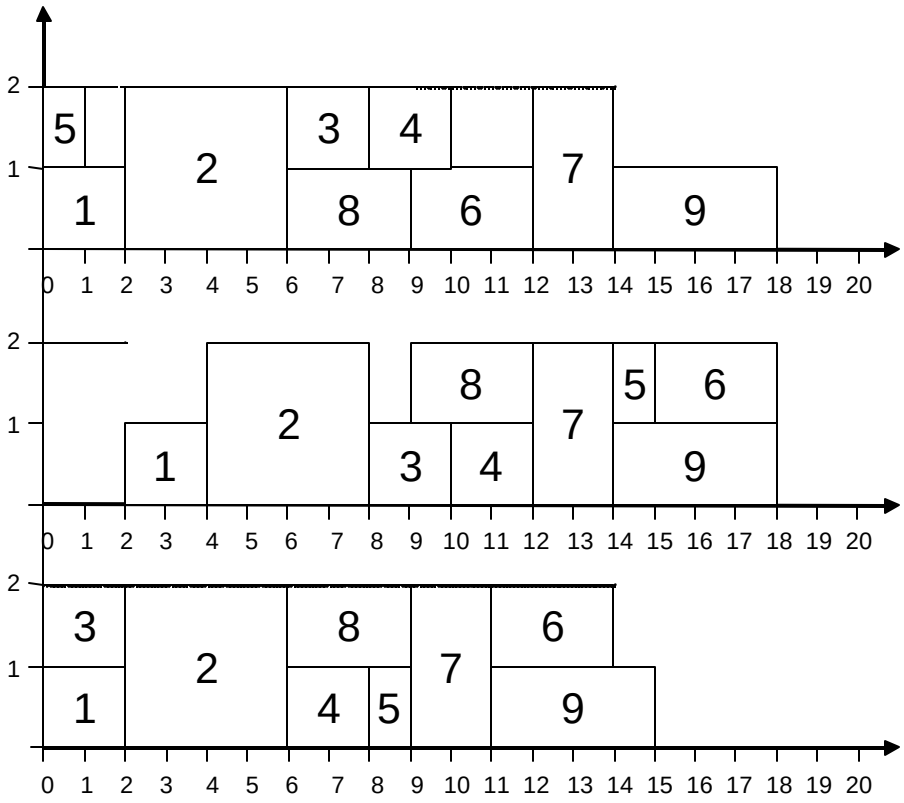

TABLE 2

Impact of parameters *m* and *b*

	***m* = 6**		***m* = 8**		***m* = 10**	
	Sum	**Avg. Dev.**	**Sum**	**Avg. Dev.**	**Sum**	**Avg. Dev.**
***b* = 1**	91,153	276%	91,149	276%	91,153	276%
***b* = 2**	91,157	276%	91,230	277%	91,261	277%
***b* = *m*-1**	91,407	278%	91,488	278%	91,513	278%

TABLE 3

Computational results

Problem Set	J30	J60	J90	J120
Sum	28,386 28,339 28,324 28,319	38,860 38,632 38,479 38,416	46,448 46,166 45,959 45,859	77,315 76,087 75,122 74,757
Avg. Dev. CPM	-	12.01% 11.35% 10.90% 10.70%	11.61% 10.93% 10.43% 10.18%	36.22% 34.07% 32.37% 31.72%
Avg. Dev. Best	0.20% 0.06% 0.02% 0.008%	1.09% 0.62% 0.32% 0.18%	1.32% 0.84% 0.50% 0.33%	3.89% 2.54% 1.49% 1.08%
Best	437 (480) 462 (480) 473 (480) 477 (480)	359 (480) 376 (480) 381 (480) 424 (480)	362 (480) 370 (480) 381 (480) 391 (480)	194 (600) 215 (600) 247 (600) 279 (600)
Avg. CPU (seconds)	0.02 0.11 1.10 10.96	0.06 0.30 3.02 30.17	0.14 0.61 6.08 60.95	0.21 1.01 10.18 102.82
Max. CPU (seconds)	0.05 0.17 1.57 14.60	0.12 0.48 4.56 46.78	0.34 1.01 10.11 100.36	0.37 1.72 15.29 155.04

TABLE 4

Comparative computational results with limit on number of schedules

Problem Set	J30		J60		J90		J120	
Author	**Dev. (%)**	**R**	**Dev. (%)**	**R**	**Dev. (%)**	**R**	**Dev. (%)**	**R**
Hartmann (1998)	0.25	5	11.89	4	–	–	36.74	5
Hartmann (2002)	0.22	3	11.70	2	–	–	35.39	2
Nonobe and Ibaraki (2002)	–	–	–	–	–	–	35.86	3
Alcaraz and Maroto (2001)	0.12	2	11.86	3	–	–	36.57	4
Bouleimen and Lecocq (2003)	0.23	4	11.90	5	–	–	37.68	6
Our procedure	0.06	1	11.35	1	10.93	1	34.07	1

TABLE 5

Comparative computational results

Problem Set	J30				J60			
Author	Sum	Dev. (%)	Avg. CPU	Max. CPU	Sum	Dev. (%)	Avg. CPU	Max. CPU
Nonobe & Ibaraki	28,337	0.06	9.07	–	38,697	11.55	26.49	–
Fleszar & Hindi	–	–	0.64	5.86	–	10.94	8.89	80.70
Valls et al. (2003)	28,335	0.06	1.61	6.15	38,671	11.45	2.76	14.61
Valls et al. (2001)	28,361	0.13	0.38	1.54	38,512	10.98	1.14	7.03
Our (5,000)	28,339	0.06	0.11	0.17	38,632	11.35	0.30	0.48
Our (50,000)	28,324	0.02	1.10	1.57	38,479	10.90	3.02	4.56
Our (500,000)	28,319	0.008	10.96	14.60	38,416	10.70	30.17	46.78
Problem Set	**J90**				**J120**			
Author	Sum	Dev. (%)	Avg. CPU	Max. CPU	Sum	Dev. (%)	Avg. CPU	Max. CPU
Nonobe & Ibaraki	46,294	11.25	181.41	–	76,600	34.99	645.33	–
Fleszar & Hindi	–	–	32.43	247.91	–	33.10	219.86	1,126.97
Valls et al. (2003)	46,247	11.12	4.63	25.49	76,356	34.53	17.00	43.94
Valls et al. (2001)	45,967	10.44	2.53	17.57	75,009	32.18	14.52	60.80
Our (5,000)	46,166	10.93	0.61	1.01	76,087	34.07	1.01	1.72
Our (50,000)	45,959	10.43	6.08	10.11	75,122	32.37	10.18	15.29
Our (500,000)	45,859	10.18	60.95	100.36	74,757	31.72	102.82	155.04